基于 Python 协程技术的 LAMOST 控制节点分布状态采集与监视系统*

田园 1,3，王锋 2,3，李建 1,3，王政 1,3，赵永恒 1,3

(1. 中国科学院国家天文台郭守敬运行与发展中心，北京 100101;

2. 广州大学，广东 广州 510006;

3. 中国科学院大学，北京 100049)

摘要：现代大型天文望远镜的控制一般由多台独立（或组成集群）的计算机来完成。这些计算机的性能、效率与可靠性直接影响整个望远镜的稳定运行。因此，需要研制一套实时对各控制计算机（节点）硬件资源信息进行采集、存储、监控的软件系统，并提供一定的预警功能。这样的系统可以有效排除隐患进而提高望远镜整体观测运行效率。本文在深入分析 LAMOST 望远镜观测运行需求的基础上，采用异步协程技术，设计和开发了一套基于 Python 语言的硬件资源监控系统软件，系统可以高效稳定地采集获取计算机各种状态，也可以获得相应部件的实时信息，并提供了多种人机交互方式，为后续开发提供了扩展接口。该系统部署于 LAMOST 环境中，实际运行表明整个系统取得了较好的效果。

关键词：资源监视；异步协程；LAMOST；大型天文望远镜运维

中图分类号：TP311.1　　文献标识码:A　　文章编号:1672-7673(2018)

大天区面积多目标光纤光谱天文望远镜(Large Sky Area Multi-Object Fiber Spectroscopy Telescope, LAMOST)也叫郭守敬望远镜是一台自主研发的主动反射施密特装置，兼顾大口径与大视场，运用并行控制技术使之可以同时跟踪多达 4000 个观测目标，是天文学界光谱获取率最高的天文望远镜[1]。郭守敬望远镜于 2008 年 10 月正式落成，到目前为止，已观测、生产和发布天文光谱超过 900 万条†。

郭守敬望远镜设计新颖，结构复杂，由观测控制系统、主动光学系统、巡天战略系统、数据处理系统等 8 个子系统构成。作为现代大型天文设备，每个子系统都离不开计算机的控制。观测期间，多达上百台的计算机组成了一个庞大的计算机集群[2]。集群中的各计算机节点(尤其是关键节点)的性能、效率和稳定性等因素直接影响整个望远镜的运行效率。对集群中各节点计算机资源的监视、预警和管理是一项繁重、困难但又必不可少的工作。

本文正是在这样的背景下开展相应的研究工作，在深入分析工程环境和工程需求的基础上，利用 Python 异步协程技术，设计和开发了一套大型望远镜计算机集群的硬件资源信息监控系统，并部署于郭守敬望远镜的工作环境。

1　需求分析与工具选择

郭守敬望远镜使用的计算机不仅数量众多，而且计算机之间的软硬件差异也十分明显。在设备类型方面，既有性能优越的大型刀片服务器、工作站，又有适应各种恶劣环境的工控机以及适合工作人员操作的台式机。在分布位置方面，数据服务式计算机(如：数据存储服务器、数据传输设备等)安置于专业机房中，人机交互式台式机安置于观测控制室，硬件驱

*基金项目：国家重点基础研究发展计划(973 计划)(2016YFE0100300);国家自然科学联合基金(U1631129, U1531132);国家自然科学基金(11603044,11403009, 11463003, 1177301)资助.

收稿日期:2018-02-13;修订日期:2018-03-22

作者简介：田园，男，助理研究员.研究方向：大型望远镜控制技术、网络通信技术. Email: tianyuan@bao.ac.cn

† http://dr5.lamost.org/

动式计算机(如相机控制集群、机架焦面控制机等)安置于望远镜主体建筑内，辅助服务式计算机(如气象站部分室外机、DIMM 等)则放置在室外。从操作系统分类来讲，目前集群中各节点计算机采用的操作系统包括了 Windows 系列、Linux（RedHat 或 Ubuntu）以及 Apple MacOS 等。

面对类型众多、位置复杂、操作系统异构的大型望远镜控制计算机集群，节点计算机的硬件资源信息监视和管理工作完全由人工逐节点实现，不但工作效率低下，而且极其容易出现错判漏判等情况，从而影响望远镜的使用。

显而易见，对于现代大型望远镜，构建一套硬件资源监控软件系统完成对集群中各种节点(尤其是关键节点)计算机的资源与状况的采集、存储、监控和分析处理是非常重要的。

在工程领域，开源的计算机集群资源监控软件主要有 Cacti、Nagios、Zabbix 等[3-5]，均在各种应用领域起到了重要作用。但是，Cacti 更倾向于单一的网络资源监视功能而不能全面地监视各节点的状态。Nagios 和 Zabbix 均可以全面监视各节点资源状态并且提供报警机制和插件模板等技术。但是由于是通用性资源监视系统，两者配置比较繁琐，再开发难度较大，难于整合到郭守敬望远镜现有的观测控制系统中。因此，基于工程需求，需要开发一套适合郭守敬望远镜的计算机集群资源监视系统。

为了保证良好的可用性和稳定性，硬件资源监控系统应采用分层方式设计和开发。综合考虑在实际工程环境中的诸多因素，系统应分为信息采集与传输层、信息接收与处理层以及信息展示与应用层。信息采集和传输层充分考虑节点之间的巨大差异，以及传输实现方式的统一性。信息的接收和处理层具备较好的接收能力和稳定的处理能力，并为上层应用提供解耦的接口。信息展示和应用层的功能应该模块化，从而达到人机交互的灵活性和友好型。整套系统还应考虑运行效率，不应给各节点以及整个网络带来较大的负载。

另外，为了更好地服务于望远镜的运行，还应该考虑将软件融入望远镜的软件体系之中。目前望远镜正在尝试采用远程望远镜系统第 2 版(Remote Telescope System 2nd version, RTS2)[‡][6]升级原有的观测控制系统 [7-8]。因此，资源监控系统需要预留接口以供 RTS2 框架使用。

基于以上工程需求分析，结合望远镜实际工作环境，采用“客户端-服务器-应用”的架构实现资源监控系统。在编程语言方面，选择通用性较强跨平台能力出众的 Python 作为主题语言，并利用 Python 的标准库(模块)以及实现各种功能的第三方库。

经过充分调研和认真筛选，文中主要涉及以下 Python 库(模块)的使用：

psutil 模块：能够轻松实现获取系统运行的进程和系统利用率（包括 CPU、内存、磁盘、网络等）信息，主要应用于系统监控、分析和管理领域。

Python 标准库的 asyncio 模块：支持异步协程。传统服务器端的设计大多采用单进程多路 IO 复用、多进程(池)、多线程(池)等技术，这些技术对提升服务器端的响应效率起到了积极的作用，但以上技术也有各自的限制和缺陷。例如：多路 IO 复用技术在编写大型服务器处理程序时过于复杂不易维护。多进程技术需要占用大量系统资源(占用内存、进程间上下文切换消耗等)，并且进程间通信实现过程较复杂。多线程技术相对于多进程技术消耗的系统资源较少，但是线程间同步、竞争、死锁等问题给服务器设计带来很大麻烦。协程是一种程序开发人员可控的用户态上下文切换技术，相比于进程、线程，协程更加轻量级且调度更加灵活，因此非常适合小数据量、逻辑简单、高并发的网络通信环境。在 Python 中，协程属于较新引入的概念。Python 从 3.4 开始支持协程，3.5 引入协程关键字(async/await)和语法，同时提供了 asyncio 标准库[§]。

curio 库：对 asyncio 模块的封装库，它进一步隐藏了事件循环和复杂回调机制，还提供了 Queue(用于协程间通信)、SingalQueue(用于系统信号处理)、run_in_process(用于子进程调

[‡] http://rts2.org
[§] https://docs.python.org/3/library/asyncio.html

度)、run_in_thread(用于子线程调度)等开发工具。从而使编写异步协程程序更加简洁安全。

PyZMQ 库：Python 版的 ZMQ 通信库。ZMQ 基于标准 socket，进一步开发了上层协议，提供了断续重传、负载均衡、发布/订阅等高级接口。适合大型软件组件间通信的解耦，使之更加稳定。

aiomysql 模块：Python 的异步 MySQL 数据库接口模块。它允许 Python 应用程序与 MySQL 数据库之间的异步数据交互，同时还提供了可配置的线程池，以提高数据库的操作性能。

2 设计与实现

2.1 总体设计

郭守敬望远镜资源监控软件系统总体分为 3 层：信息采集器(客户端)、中央信息处理服务器以及上层应用模块，如图 1。

部署在郭守敬望远镜计算机集群各节点计算机上的信息采集器(客户端)根据默认设置和配置文件确定信息采集间隔时间以及本节点需要采集的具体信息内容后，周期性地采集本节点硬件资源信息并发送给中央信息处理服务器。

中央信息处理服务器接收众多客户端发来的信息，进行信息处理(如：数据验证、数据分析、数据存储等)，并通过自身的服务接口向上层应用模块提供数据服务。

上层应用模块可以有多个，分别实现不同的功能(如图形界面人机交互、网页显示、接入观测控制系统、数据详细分析统计等)。它们均通过中央信息服务器的数据提供接口获取数据，然后这些数据完成自己的工作。

应用模块和中央信息处理服务器相互独立，不规定启动先后顺序，也不需要在同一台电脑上运行，最大程度实现系统解耦，从而提高系统的稳定性和可用性。

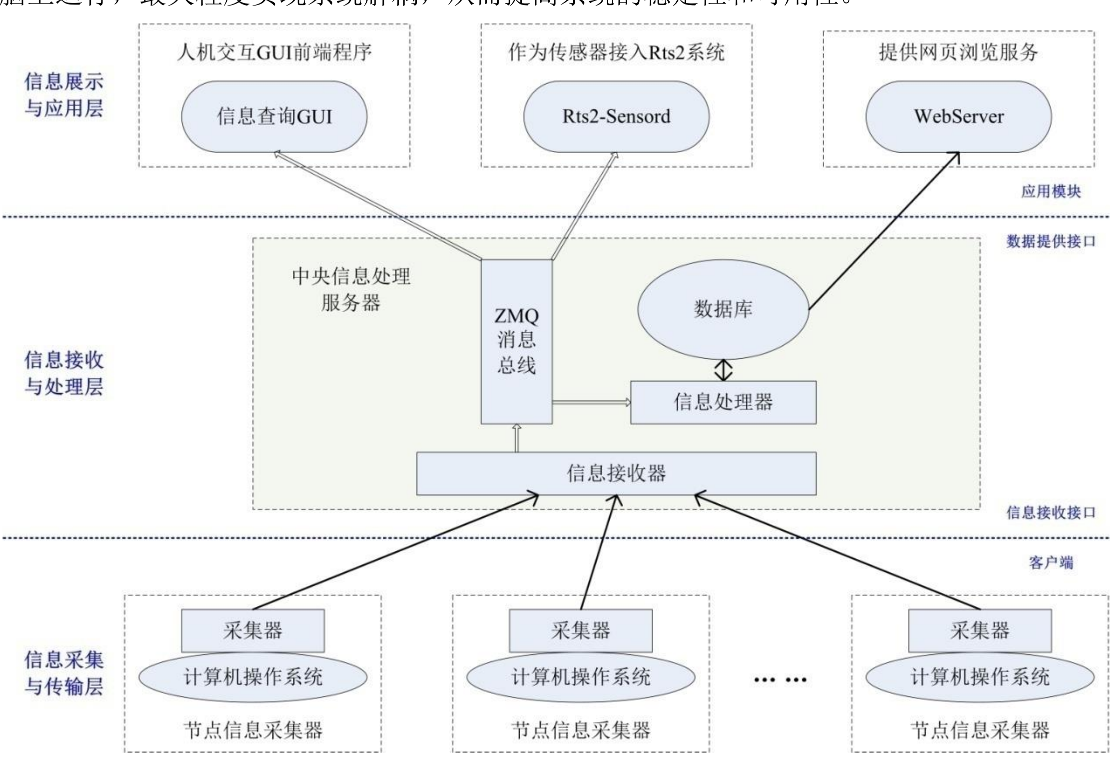


图 1 郭守敬望远镜资源监控系统总体框架

Fig. 1 The framework of Monitoring System for LAMOST resource

2.2 客户端

客户端主要责任是周期性采集节点系统资源信息，将信息发送给服务器。

考虑代码的简洁性和易维护性，选取 Python 的 psutil 包进行信息采集。绝大多数节点计算机的操作系统自带 Python 解释器以及相应包。对于未安装 psutil 包的节点采用 pip 安装

或源码安装也十分简易。极个别节点无法安装 Python，则采用 C 语言调用原生系统库接口获取信息。如下代码为客户端采用 psutil 包获取系统资源的部分代码。

```
import psutil as ps
… …
cpu = ps.cpu_percent(interval=60)            # 获取CPU使用率，周期60秒
mem = ps.virtual_memory()                    # 获取内存使用率
disk = ps.disk_usage('/')                    # 获取挂载点'/'所对应磁盘分区使用情况
nic_in = ps.net_io_counters().bytes_recv     # 获取网卡接收字节数
nic_out = ps.net_io_counters().bytes_sent    # 获取网卡发送字节数
procs = ps.pids()                            # 获取进程列表
… …
```

为保证信息传输的通用性并尽量避免在各节点计算机上安装额外软件库或包，客户端信息发送采用标准的 TCP 通信协议，通信功能采用 Python 实现：设置错误容忍阈值，将采集信息和发送信息功能包裹在循环体中。出现通信故障时捕获异常错误计数器自增 1，若错误计数器未到阈值则休眠指定时间后重新循环，若已到达阈值则结束程序释放系统资源。

2.3 服务器

中央信息处理服务器是整套系统的枢纽，主要任务有两个：（1）接收大量客户端周期性发来的资源信息；（2）对信息进行处理后存入 MySQL 数据库。因此，对其运行效率和稳定性要求很高。为了使设计思路清晰、实现代码简洁，实际方案采用了两个子进程，分别为异步消息接收器子进程和 MySQL 异步处理器子进程。中央信息处理服务器内部结构如图 2。

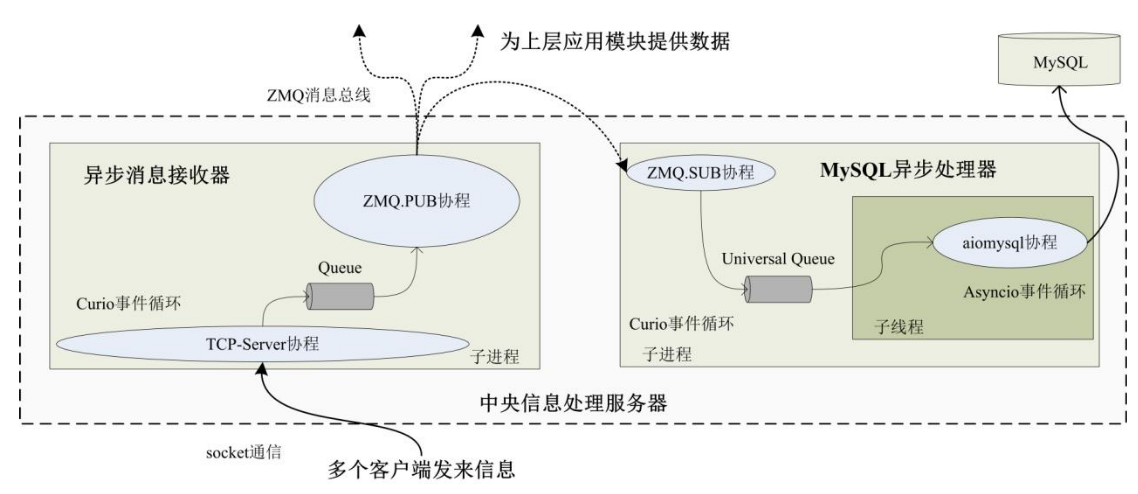


图 2 中央信息处理服务器内部结构

Fig. 2 The internal structure of the central message handle server

异步消息接收器子进程内部包含 TCP-Server 协程和 ZMQ.PUB 协程。TCP-Server 协程负责处理节点采集器(客户端)的 TCP 连接申请以及并发地接收多客户端发来的消息。ZMQ.PUB 协程负责将接收的消息发布到消息总线上，以供上层应用模块使用。两者之间采用 curio 库的 Queue 实现单线程内不同协程之间的异步通信，整个子进程由 curio 库事件循环驱动。

MySQL 异步处理器子进程内包含 ZMQ.SUB 协程和 aiomysql 协程。ZMQ.SUB 协程负责向消息总线订阅并获取消息。aiomysql 协程负责将消息异步地存储到 MySQL 数据库。由于 aiomysql 模块必须采用标准的 Python.asyncio 事件循环驱动(curio 库目前尚未直接提供 aiomysql 模块接口)，而 Python.asyncio 事件循环本身必须独占一个线程，因此程序在实现过程中需要为 aiomysql 协程创立独立的子线程。两者之间采用 curio 库的 Universal Queue 实现异步通信。与异步消息接收器中使用的 Queue 不同，curio 库提供的 Universal Queue 可以解决分属于多个线程的不同协程之间的异步通信问题。整个子进程由 curio 库事件循环驱动。

此外，中央信息处理服务器采用 ZMQ 的发布/订阅机制组建消息总线，实现对上层应用

模块提供数据的功能。ZMQ.PUB 端口用于发布消息，可以有任意数量的 ZMQ.SUB 端口用于订阅和接收消息。ZMQ.PUB 和 ZMQ.SUB 无需在同一计算机上部署，且运行互补干扰。由于 ZMQ 以上优点，极大地简化了服务接口层的开发难度，实现了中央信息处理服务器与各应用模块之间的完全解耦。

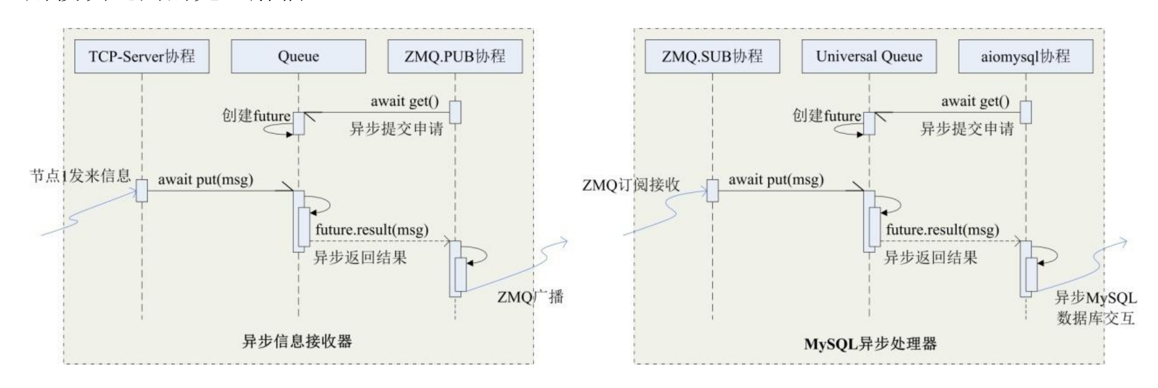


图 3 中央信息处理服务器内部时序图

Fig. 3 The internal sequence diagram of the central message handle server

中央信息处理服务器内部组件时序图如图 3。由于组件均为协程，组件间均采用异步方式通信，协程逻辑执行与上下文切换由用户精确控制，因此在不占用更多操作系统资源的前提下，极大地提高了服务器的并行处理能力和数据的吞吐量。

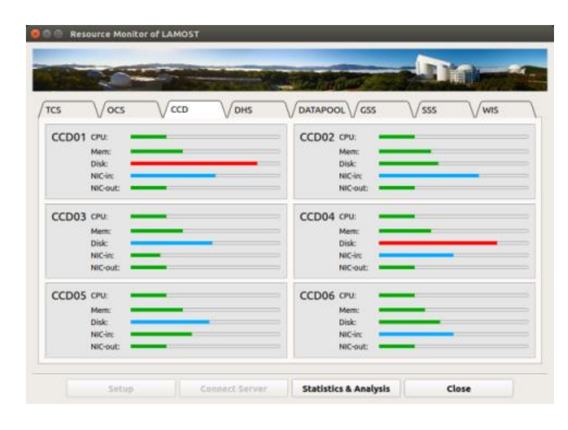
实际运行中，中央信息处理服务器采用 Linux 后台守护进程方式运行，并设置为开机自启动，从而保证了程序运行更加稳定和高效。

2.4 应用模块

应用模块主要负责数据的应用与展示。资源监控系统允许运行多个相互独立的应用模块完成不同的功能。由于 ZMQ 消息总线的应用，使中央信息处理服务器与应用模块之间彻底解耦，因此，极大地降低了各种应用模块的开发难度。

根据工程需求，目前资源监控系统提供 3 种应用模块：基于 PyQt 的图形界面模块、基于 RTS2 系统的传感器设备(Rts2-sensord)模块和基于 Django 的网站服务模块。

图 4 为本机图形界面模块运行截图。图形界面实时显示各节点计算机资源信息并以颜色进度条的方式为运维工程师提供资源占用情况。图 4 中，01 号和 02 号 CCD 计算机的硬盘占用率超过 80%，因此，图形界面相应节点子界面的硬盘进度条变成红色，以提醒观测者注意，从而达到预警效果。



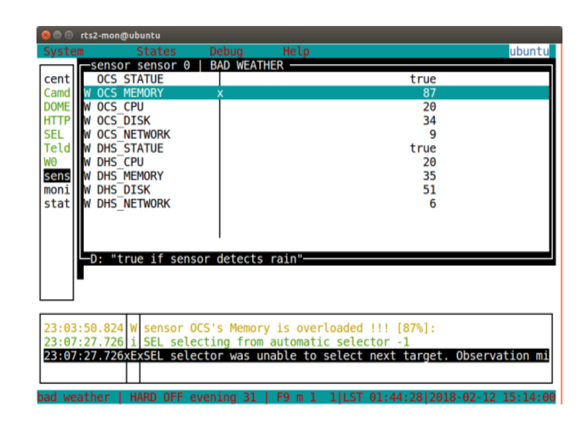
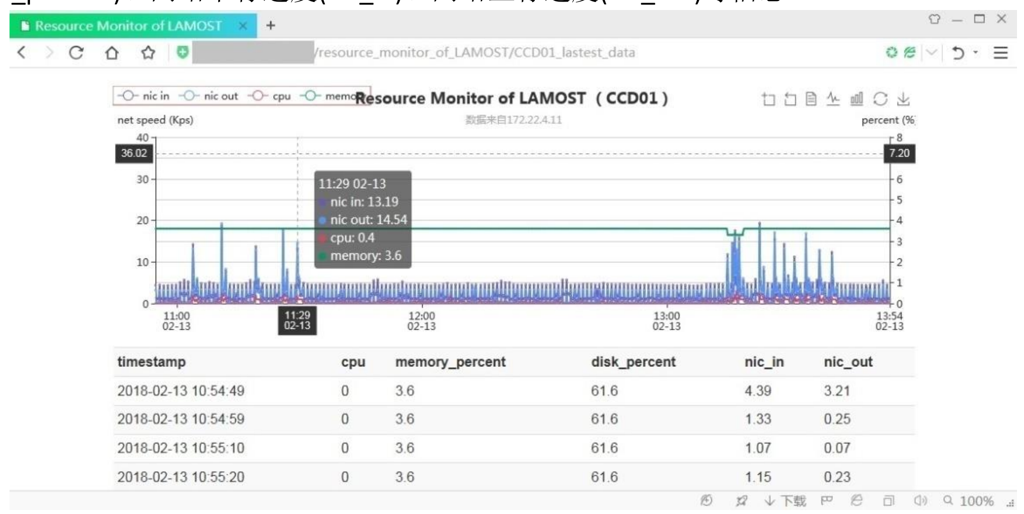


图 4 图形界面模块截图　　图 5 RTS2 系统监视器截图

Fig. 4 The screenshot of GUI module　　Fig. 5 The screenshot of RTS2 monitor

资源监控系统为 RTS2 框架预留了传感器模块 RTS2-Sensord。将其接入 RTS2 系统后，RTS2 系统便可以收到资源预警信息。图 5 是 RTS2 系统监视器收到预警信息的截图。

架设于郭守敬望远镜内网中网站模块提供网页浏览查询服务，允许观测者和运维工程师在内网中的计算机上通过浏览器查询各节点资源信息数据。图 6 为 01 号 CCD 计算机的资源信息页面。其中上半部分已曲线图的方式显示了该节点计算机各种资源信息的历史记录和趋势，并以可设置虚线的形式提供预警功能。下半部分为该节点信息列表，表中分别显示了信息采集时间(timestamp)、CPU 使用率、内存占用率(memory_percent)、硬盘使用率(disk_percent)、网络下行速度(nic_in)、网络上行速度(nic_out)等信息。



timestamp	cpu	memory_percent	disk_percent	nic_in	nic_out
2018-02-13 10:54:49	0	3.6	61.6	4.39	3.21
2018-02-13 10:54:59	0	3.6	61.6	1.33	0.25
2018-02-13 10:55:10	0	3.6	61.6	1.07	0.07
2018-02-13 10:55:20	0	3.6	61.6	1.15	0.23

图 6 网页服务

Fig. 6 Web Service

3 讨论

3.1 系统自身开销与网络负载

资源监控系统软件运行自身也需要消耗部分系统资源，因此在软件设计和开发过程中，在保证可用性的前提下应建立简化和优化代码。将资源监控系统部署在实际工程环境中，实际运行测试结果如下：

节点消息采集器(客户端)资源消耗：CPU 占用不超过 0.2%，内存不超过 4KB。

```
lamost@localhost:~
[lamost@localhost ~]$ ps -e -o'pid,cmd,pcpu,rsz' | grep rsmolc.py | grep -v grep
24622 python2 ./rsmolc.py start    0.0  3768
```

中央信息处理服务器资源消耗：CPU 不超过 0.5%，内存不超过 40MB。

```
lamost@localhost:~
[lamost@localhost ~]$ ps -e -o'pid,cmd,pcpu,rsz' | grep resource_mon | grep -v grep
12354 python3 ./resource_monitor_  0.0  9648
12355 python3 ./resource_monitor_  0.0 10624
12356 python3 ./resource_monitor_  0.0 11308
```

可见，本系统不会对节点计算机和中央信息服务器产生过大的资源消耗。

而网络传输方面，目前单节点信息采集器产生的信息以字符串形式传输，单条信息不超过 128 个字节，信息采集周期默认值为 10 s。因此，100 个节点采集器每秒产生的信息大小不超过 1280 字节，即整个系统对内网的带宽占用不超过 1.25KBps。未来，如果由于单节点信息量增多或系统总节点数增加导致网络带宽占用过多，可以考虑放弃字符串采用二进制数据传输。

因此，郭守敬望远镜资源监控系统无论是在自身资源消耗方面还是网络负载方面，都符合当前工程需求。

3.2 网络通信架构：集中式与分布式的选择

目前，网络通信主流架构主要分为集中式设计和分布式设计两大类。集中式设计需要有强大健壮的中央服务器作为整个通信系统的枢纽。优点是技术成熟，且便于管理和维护。缺点是依赖于中央服务器的稳定性和处理能力。而分布式则是信息或功能分散在不同的计算机

上，节点间协作共同完成任务，因此对每台计算机的性能要求不高。优点是可以极大地降低服务器的成本。缺点是程序设计更复杂，协作通信量增加。

具体到本项目，虽然涉及到集群中的计算机数量众多，但通信量和数据处理量均很小，因此选择集中式设计，即“客户端-服务器-应用”模式作为整体架构更为合理，从而避免了分布式中的数据同步等问题，降低开发和维护难度。

3.3 客户端与服务器之间通信协议的选择

在采集到信息之后，客户端需要通过网络将信息发送给服务器。网络信息传输的实现可以选择，例如：UDP 传输、TCP 传输、ACE 通信库、ZMQ 通信库等。采用 UDP 通信可靠性相对较差，因此暂不考虑。而 ACE 通信库是一套完善的 C++通信库，它完全采用面向对象设计，支持和采用多种设计模式。但是由于过于庞大复杂且开发维护难度很高，因此并不适合本文提到的实际工程需求。采用 ZMQ 通信方式(见服务器内部通信实现章节)虽然使用方便灵活但是需要在各个客户端安装 ZMQ 库。如前文所述，各个节点硬件设备差异较大，操作系统种类繁多，如果根据各节点软硬件配置选择相应版本的 ZMQ 库并进行安装，不但工作量巨大、可操作性很差，而且不利于今后设备更新和扩展。而 TCP 传输作为最常见的底层通信协议，各种操作系统均自带了其实现库，无需再自行安装配置。因此，客户端与服务器之间的通信采用 TCP 通信协议。

3.4 系统的预警和分析处理功能扩展

郭守敬望远镜资源监控系统已初步完成，部署于实际工程环境中能够准确有效地采集和存储集群中各节点计算机资源信息数据。但仍有以下两方面不足：

（1）在人机交互预警方面，如 2.4 节所述，目前通过本机图形界面资源条颜色 (绿、蓝、红) 变化，以及网页预警线的方式为工作人员提供最基本提醒功能。未来可以借鉴 Zabbix 系统的实现思路根据预警信息类型或错误等级为工作人员提供邮件推送、微信消息等功能，从而进一步提高系统的友好性。

（2）已存储的资源信息数据的后期分析、统计和处理功能目前尚未完成。主要原因在于该功能需要不断积累大型望远镜运行维护经验。

不过，由于服务器内部采用了基于发布/订阅机制的 ZMQ 消息总线技术，使预警组件和数据分析处理组件与整个软件框架完全解耦，极大地降低了进一步开发的技术难度，因此，资源监控系统会不断完善，最终为望远镜高效运行提供有力的保障。

4 结束语

郭守敬望远镜是由我国自主研发的目前世界上光谱获取率最高的大型天文望远镜。其内部管理与控制计算机庞大、节点众多、种类繁杂，本系统的设计，实时采集、存储和分析各节点计算机硬件资源信息，有效地提高望远镜的运行稳定性和观测效率。系统运行表明，本系统的设计与研发是成功的，同时这样的模式也为其他大型望远镜的观测控制与后期维护提供有价值的参考。

参考文献：

[1] 赵永恒. 天文望远镜的自动观测技术 [J]. 科研信息化技术与应用，2012，3(4)：11-16.
Zhao Yongheng. Technology of automatic observation of astronomical telescope [J]. E-Science Technology & Application，2012，3(4)：11-16.

[2] 罗阿理，田园，宋静，等. LAMOST 观测控制系统设计与实现[J]. 科研信息细化技术与应用，2012，3 (4)：76-85.
Luo Ali，Tian Yuan，Song Jiang，et al. Design and implementation of LAMOST observatory control system [J]. E-Science Technology & Application，2012，3(4)：76-85.

[3] 赵文瑞，卢志刚，姜政伟，等. 网络安全综合监控系统的设计与实现[J]. 核电子学与探测技术，2014，4(4)：419-424.
Zhao Weirui，Lu Zhigang，Jiang Zhengwei，et al. The Design and Development of Network Security Comprehensive Monitoring System[J]. Nuclear Electronics & Detection Technology，2014，4(4)：419-424.
[4] 郭晓慧，李润知，张茜，等. 基于 Zabbix 的分布式服务器监控应用研究[J]. 通信学报，2013，9(Z2)：94-98.
Guo Xiaohui，Li Runzhi，Zhang Qian，et al. Application research on distributed Zabbix network monitoring system[J]. Journal on Communications，2013，9(Z2)：94-98.
[5] 赵哲, 谭海波, 赵赫, 等. 基于 Zabbix 的网络监控控系统[J]. 计算机技术与发展, 2018, 1(1)：144-149.
Zhao Zhe, Tan Haibo, Zhao He, et al. Network Monitoring System Based on Zabbix[J]. Computer Technology and Development, 2018, 1(1)：144-149.
[6] 范玉峰，辛玉新，白金明，等. 丽江站 BOOTES-4 综述[J]. 天文研究与技术，2015，12(1)：78-88.
Fan Yufeng，Xin Yuxin，Bai Jinming，et al. An overview of the BOOTES-4 at the Lijiang obervatory[J]. Astronomical Research & Technology，2015，12(1)：78-88.
[7] 李建，崔辰州，赵永恒，等. 基于 RTS2 的 RAO 管理系统二次开发[J]. 天文研究与技术—国家天文台台刊，2013，10(3)：264-272.
Li Jian，Cui Chenzhou，Zhao Yongheng，et al. A secondary development of an RAO management system based on the Remote Telescope System-2nd version[J]. Astronomical Research & Technology—Publications of National Astronomical Observatories of China，2013，10(3)：264-272.
[8] 卫守林，陈亚杰，梁波，等. RTS2 中新 CCD 类型扩展方法[J]. 天文研究与技术—国家天文台台刊，2014，11(3)：281-286.
Wei Shoulin，Chen Yajie，Liang Bo，et al. Methods of constructing new CCD classes in the RTS2[J]. Astronomical Research & Technology—Publications of National Astronomical Observatories of China，2014，11(3)：281-286.

LAMOST resource monitoring system based on Python Coroutines Technique

Abstract：The control of modern large-scale astronomical telescopes is generally accomplished by multiple independent (or clustered) computers. The performance and efficiency of each node computer directly affect the operational stability of the whole telescope. A source monitor software system is an import and essential management tool for the operation and maintenance of large telescopes. This system can collect and store the hardware resource information of each node, and it can further monitor and analyze this information. When necessary, the system will provide the observers with warning messages and suggestions in order to eliminate hidden dangers and to improving the observational efficiency of the overall telescope. Based on the deep analysis of the engineering requirements, we design and develop a resource monitor system for LAMOST. The system is implemented using Python language and

asynchronous coroutines technology, and provides a variety of human-computer interactive functional modules and extension interfaces. We has deployed it in the actual project environment and achieved rather effective results. The work of this paper provides rather valuable references for managing and maintaining other large telescope.

Key words: Resource monitor; Asynchronous coroutines; LAMOST ; Telescope operations